ANL/NE-16/6

SHARP User Manual

Nuclear Engineering Division

About Argonne National Laboratory
Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (http://www.osti.gov/scitech/)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

ANL-NE-16/6

SHARP User Manual

March 31, 2016

prepared by
Y. Q. Yu, E. R. Shemon, and J.W. Thomas
Nuclear Engineering Division, Argonne National Laboratory

V. Mahadevan, R. Rahaman
Math and Computer Science Division, Argonne National Laboratory

J. Solberg
Methods Development Group, Lawrence Livermore National Laboratory

ABSTRACT

SHARP is an advanced modeling and simulation toolkit for the analysis of nuclear reactors. It is comprised of several components including physical modeling tools, tools to integrate the physics codes for multi-physics analyses, and a set of tools to couple the codes within the MOAB framework. Physics modules currently include the neutronics code PROTEUS, the thermal-hydraulics code Nek5000, and the structural mechanics code Diablo. This manual focuses on performing multi-physics calculations with the SHARP ToolKit. Manuals for the three individual physics modules are available with the SHARP distribution to help the user to either carry out the primary multi-physics calculation with basic knowledge or perform further advanced development with in-depth knowledge of these codes.

This manual provides step-by-step instructions on employing SHARP, including how to download and install the code, how to build the drivers for a test case, how to perform a calculation and how to visualize the results. Since SHARP has some specific library and environment dependencies, it is highly recommended that the user read this manual prior to installing SHARP. Verification tests cases are included to check proper installation of each module. It is suggested that the new user should first follow the step-by-step instructions provided for a test problem in this manual to understand the basic procedure of using SHARP before using SHARP for his/her own analysis. Both reference output and scripts are provided along with the test cases in order to verify correct installation and execution of the SHARP package. At the end of this manual, detailed instructions are provided on how to create a new test case so that user can perform novel multi-physics calculations with SHARP. Frequently asked questions are listed at the end of this manual to help the user to troubleshoot issues.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 Introduction of SHARP

1.1 Introduction on SHARP Multi-physics Code System and Coupling Methodology

SHARP [1], developed under the NEAMS program, is an advanced modeling and simulation toolkit for the analysis of nuclear reactors. SHARP is comprised of several components, including physical modeling tools, tools to integrate the physics codes for multi-physics analyses, and a set of tools to couple the codes within the MOAB [2] framework. Physics modules currently include the PROTEUS [3] neutronics code, the Nek5000 [4] thermal-hydraulics code, and the Diablo [5] structural mechanics code. The development philosophy for the physics modules is to incorporate as much fundamental physics as possible, rather than developing tools for specific reactor analysis applications. This empowers designers to analyze transformative reactor concepts with simulation tools that are not limited to available experimental data sets from currently existing reactor designs. By developing the tools to be highly efficient on parallel computing platforms; employing millions of processor cores; engineering-scale simulations become practical on high-performance computers currently available at the DOE complex. Development efforts strive to work in tandem with efforts in experimentation, so that the tools are validated to produce accurate results for modeling physical phenomena that have been identified as important for nuclear reactor analysis. By taking this approach, SHARP supports nuclear reactor analysis and design activities for DOE programs and industrial partnerships with trustworthy modeling and simulation tools.

In order to produce a fully coupled physics simulation capability, two obvious approaches can be pursued. In one approach, existing single-physics codes/components can be assembled into an overall coupled simulation code with appropriate interfaces to communicate between the components to capture the nonlinear feedback effects. This is generally referred to as a "small-f" or "bottom-up" framework approach [1, 6]. The other approach is to use an integrated, coupled-physics modeling framework, with new code pieces for each relevant physics area developed inside that framework from scratch. This is sometimes referred to as a "large-F" or "top-down" approach [7, 8]. The primary advantage of the former approach is that it preserves several man-years invested in existing verified and validated individual physics modeling codes, but at the cost of some intrusive modifications to enable the software interfaces. The large-F approach avoids intrusive interfacing by providing a unified platform to enable coupling, but at the cost of re-writing all the necessary physics codes and verifying the components individually and as a whole. The overall approach being pursued in the NEAMS

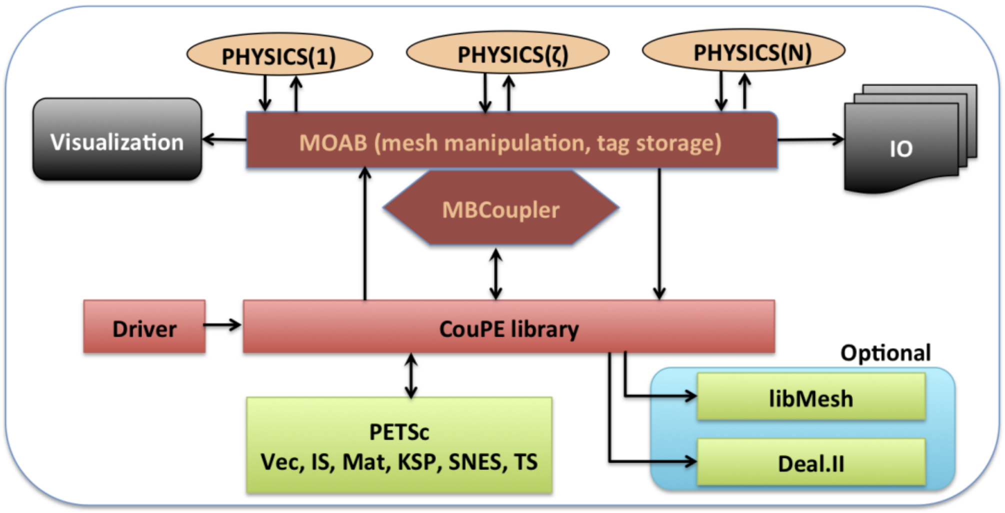
Reactor Product Line (RPL) effort is to develop and demonstrate a small-f framework for performing coupled multi-physics analysis of reactor core systems. This system takes advantage of many single-physics codes also sponsored by the overall NEAMS program over past several years.

In the SHARP framework, MOAB interfaces are implemented for 3 different physics components that are relevant to fast reactor physics analysis. The addition of a new physics component to the framework requires integration and ability to read the mesh and possibly associated data from iMesh/MOAB formats, along with implementation to propagate solution variables back onto the mesh after their computation via tags defined either on discrete vertices or elements. Because of the various storage formats used in physics models, and the parallel domain-decomposed environment in which these calculations are usually run, this integration process can be somewhat involved.

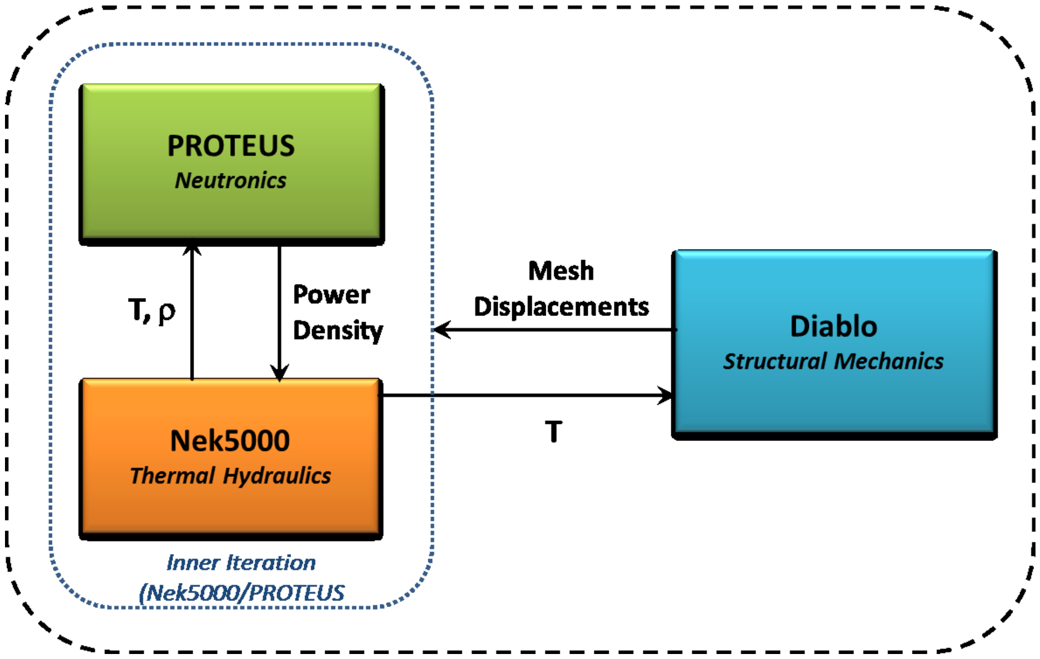
A multi-physics reactor core modeling code can be constructed in many ways, and numerous past efforts have provided stepping-stones for future efforts [8]. What distinguishes the SHARP effort from others is the goal of flexibility in the physics, discretization types, and software options supported by the framework. This section describes the SHARP modeling approach in detail and illustrates how various existing physics codes have been connected to this framework.

As stated above, SHARP employs a “bottom-up” approach, so it can use existing physics codes and take advantage of existing infrastructure capabilities in the SIGMA framework, including the MOAB mesh database, the Coupled Physics Environment (CouPE), which utilizes the widely used, scalable PETSc library [**Error! Reference source not found.**]. Using an existing physics code in this system (Figure 1.1) requires that the system support the mesh type used by the individual physics models. The physics models can retain their own native representation of the mesh, which gets transferred to and from MOAB’s representation through a mesh adaptor; or it can use MOAB’s representation directly.

In practice, this means that the coupled system may be solved on multiple meshes, each of which models part or the entire physical domain of the problem. To perform efficient coupled calculations, the results must be transferred from the mesh on which they are generated (source mesh), to the mesh for which they provide initial or boundary conditions (target mesh) due to nonlinearity introduced because of coupling between physics models.



(a)



(b)

Figure 1.1. (a) Depiction of the "bottom up" multi-physics coupling approach provided by SIGMA, and (b) Sketch of the SHARP global iteration strategy.

1.2 Neutronics Module: PROTEUS

PROTEUS is a set of neutronics solvers developed at Argonne National Laboratory for solving nuclear reactor applications, including discrete ordinates, method of characteristics, and nodal methods. SHARP currently couples to the PROTEUS-SN code (also known as SN2ND), a high-fidelity deterministic neutron transport solver based on the second-order even-parity formulation of the transport equation [9]. For simplification of terminology, we refer to PROTEUS-SN as PROTEUS in this document.

The application scope targeted for PROTEUS ranges from the homogenized assembly approaches prevalent in current reactor analysis methodologies to explicit geometry approaches, with the ability to perform coupled calculations to thermal-hydraulics and structural mechanics. The PROTEUS solver has a proven capability of using existing petascale parallel machines to solve problems with demonstrated scalability of over 70% (strong scaling) at over 250,000 processors (on BlueGene/P). These achievements of PROTEUS were made possible by partitioning the space-angle system of equations over the available processors and utilizing established iterative solution techniques from the neutron transport community combined with the parallel algorithms in the PETSc toolbox.

Interfaces to the SIGMA framework have been written to handle PROTEUS meshes that describe detailed geometries with multiple blocks (collections of elements, each with uniform material properties) with appropriate specification hooks for temperature-dependent material cross-section evaluation and interpolation. This interface is essential to capture the nonlinear feedback effect from thermal-hydraulics. Additionally, each region can be assigned a material model which further allows PROTEUS to compute the density of various isotopes based on temperature or other parameters. (Currently the only material model supported specified the recomputation of sodium density based on temperature). When structural mechanical feedback is present, causing mesh deformation, PROTEUS can use the new mesh as well as automatically recalculate material density changes (thereby affecting cross-sections) to enable direct coupling to a deformation code such as Diablo.

The eigenvalue solver in PROTEUS computes the neutron flux shape, computes the power distribution in the reactor, and then places the computed data in appropriate SIGMA mesh tags.

The power solution field is then propagated to the other physics solvers via the data-coupling interfaces that support tight coupling with thermal-hydraulics, which uses the tag data as a thermal source term to compute temperature fields. Several verification studies have been performed during the quality assurance process to ensure that the coupled solver solutions are physically meaningful. The PROTEUS-SN Methodology Manual and PROTEUS-SN User Manual are available in the PROTEUS module documents directory as well as online. PROTEUS-SN training material is available upon request.

1.3 Thermal Hydraulic Module: NEK5000

The Nek5000 computational fluid dynamics solvers are based on the spectral element method developed by Patera [11]. Nek5000 supports two different formulations for spatial and temporal discretization of the Navier-Stokes equations. The first is the P_N-P_{N-2} method with velocity/pressure spaces based on tensor-product polynomials of degree N and N-2 respectively. The second is the low-Mach number formulation of Tomboulides and Orszag [12], which uses consistent order-N approximation spaces for both the velocity and pressure. The low-Mach number formulation is also valid at the zero-Mach (incompressible) limit [13]. The Nek5000 code has been extensively verified and validated for several benchmark problems and has a proven scalability in existing petascale architectures up to 131,072 processors (over a billion degrees-of-freedom). The NEK5000 User Manual and NEK5000 Training Material are available on line.

1.4 Structural Mechanics Module: Diablo

The Diablo code being developed at Lawrence Livermore National Laboratory uses implicit, Lagrangian finite-element methods for the simulation of solid mechanics and multi-physics events over moderate to long time frames [5]. A primary focus is nonlinear structural mechanics and heat transfer. The code provides a venue for applying parallel computation to discretization technologies developed and user-tested in the legacy serial-processor codes NIKE3D and TOPAZ3D. Diablo is built around Fortran 95 data structure objects and a message-passing programming model. The architecture provides flexibility for the addition of other field problems, such as electromagnetics.

In structural analysis of mechanical assemblies, a key functionality is "contact": capturing the interaction between unbonded material interfaces. The Diablo team has broad experience with contact problems and has created state-of-the-art algorithms for their solution. Their experience with contact motivates the use of low-order spatial discretizations, such as eight-node hexahedra for continua and four-node quadrilaterals for shells. Appropriate formulations are employed to accommodate nearly incompressible material models, such as for metal plasticity and rubber elasticity. Global algorithms include second-order and quasi-steady time integration and a number of approaches for nonlinear iteration: full Newton, modified-Newton, multiple quasi-Newton updates, and line search. Linear solvers are utilized from multiple libraries. The Diablo User Manual is available in the SHARP distribution package.

2 SHARP Configuration and Installation

2.1 Access

The SHARP ToolKit is distributed within the U.S. under a government-use license. For access, please contact sharp-dev@mcs.anl.gov. Users will receive a tarball of the latest SHARP release package. The release package contains everything necessary to run a multi-physics reactor problem with SHARP: the configuration scripts, physics module source code, SHARP drivers, single physics and coupled physics verification problems.

2.2 Basic Installation Requirements

Before beginning to install SHARP, the user must have available recent versions of the Intel compilers (recommend 13+ although older versions may work). If not using Diablo, the GNU compilers (5.x) can alternatively be used. A recent version of the autotools toolchain (autoconf v2.63 or higher; and automake v1.11 or higher) must also be installed on the system before proceeding.

2.3 Module and Package Dependencies

At a minimum, the SHARP toolkit must be built with the Nek5000, PROTEUS, and the SIGMA driver. Diablo is disabled by default and, if desired, may be enabled during configuration (see "Advanced Configuration Options," below). Required and optional third-party packages are listed in Table 2.1. Required versions are noted.

Table 2.1. List of SHARP toolkit dependencies

Package	Diablo disabled	Diablo enabled	Version
BLACS		X	
BLAS/LAPACK		X	
EXODUS II		X	6.06
HDF5	X	X	1.8
HYPRE		X	2.9
Metis	X	X	5.1
MILI		X	13.1
MOAB	X	X	4.7
MPI	X	X	2
MUMPS		X	4.10
NetCDF		X	4.3
ParMetis		X	4.0

PETSc	X	X	3.1
SCALAPACK		X	
SILO		X	4.9

2.4 Basic Configuration Options

SHARP's configuration tools handle the entire configuration/compilation process, including all third party libraries. The SHARP package can be installed out of the box with minimal configuration options. If the user wishes to have SHARP install all required dependencies, the "--download-essential" configuration option is used. If the user wishes to use a pre-installed MPI library with compiler wrappers, the MPI_DIR=<path/to/mpi> flag should be passed, otherwise, MPI will also be installed. Diablo is disabled by default and can be turned on with the configuration option "--enable-diablo".

The exact steps to a basic installation procedure are outlined below as an example.

Step 1:

Copy the tarball to your working directory and extract the files:

```
$cp SHARP.tar.gz /home/me/working/dir/
$cd /home/me/working/dir/
$tar -xzf SHARP.tar.gz
$cd SHARP
```

For developers, checkout the source from the SVN repo and change directories:

```
$cd /home/me/working/dir/
$svn co https://svn.mcs.anl.gov/repos/SHARP/trunk SHARP
$cd SHARP
```

For the purpose of clarifying this document, we will define an environmental variable SHARP_DIR to represent this root-level of the SHARP distribution package. That is, SHARP_DIR is the path that would be set if, hypothetically, you were to issue the following command at this point:

```
$export SHARP_DIR=$PWD
```

Note, however, that it is not necessary for users to create this variable, and SHARP will not look for it in the user's environment. If the above commands were literal, then SHARP_DIR would be set to /home/me/working/dir/SHARP.

Step 2: To configure SHARP, the user first needs to run a top-level configuration generator script, aptly named “bootstrap”. This script will verify whether the system contains the necessary and supported version of autotools before proceeding further. After verification, the autotools toolchain (aclocal, autoheader, autoconf, automake) can be used to generate the configuration script. An example output from successfully running the bootstrap script is shown below.

```
$./bootstrap

-----------------------------------------------------------------
Bootstrap for SHARP build system.
Beginning to run bootstrap in /home/me/working/dir/SHARP.
-----------------------------------------------------------------
Scanning dependencies...
Checking for autoconf........ [ found version 2.69 ]
Checking for autoheader........ [ found version 2.69 ]
Checking for aclocal........ [ found version 1.14.1 ]
Checking for automake........ [ found version 1.14.1 ]
Checking for libtoolize........ [ found version 2.4.4 ]
Checking for autoreconf........ [ found version 2.69 ]
--------------------------------------------------------------------
Found all necessary dependencies. Proceeding with setup...
--------------------------------------------------------------------
Running the autotools...
Running autoreconf..... [ done ]
--------------------------------------------------------------------
Done bootstrapping your system. You may now run ./configure.
To see options use ./bootstrap --help or view the README file.
--------------------------------------------------------------------
```

Step 3: Create a new “build” directory, configure and build SHARP with desired set of features. For example, the user may quickly set up and running on ANL’s LCRC Blues Cluster with the following commands:

```
On the ANL cluster Blues:
$cd $SHARP_DIR
$mkdir build
$cd build
$export MPI_DIR=/soft/mvapich2/2.0-intel-13.1
$../configure --enable-diablo --with-mpi=$MPI_DIR --download-
essential
$make all
```

```
On workstations in the ANL MCS Division:
$cd $SHARP_DIR
$mkdir build
$cd build
$ ../configure --enable-diablo --download-essential --with-
mpi=/soft/apps/packages/mpich2-1.4.1p1-intel –prefix=$SHARP_DIR
FFLAGS=-lifcore FCFLAGS=-lifcore
$make all
```

Step 4: (Optional): The open source code VisIt is necessary for results visualization. The user should check with his/her local system administrator whether VisIt has already been installed. If not, the user can install it by downloading the source code, and running the "build_visit" script with the following command in any directory that user wants to install VisIt:

```
$ <script name> --itaps --netcdf --hdf5 --szip --console --silo --
python
```

The download and installation process for VisIt requires approximately 3 hours.

2.5 Advanced Configuration Options

Detailed instructions on configuration and installation can be found in the README file distributed with SHARP. The vast majority of users will find it convenient to use SHARP's most basic configuration commands listed in the previous section to download all essential dependencies. Otherwise, several families of configuration options are available to control whether certain features are enabled/disabled, or if certain dependencies (pre-installed) need to be used in the current build or if the user wants the package manager to auto-download and configure some of the dependencies.

2.5.1 Enabling/disabling compile-time features or packages

Compile-time features are enabled by options of the form:

```
--enable-<feature>[=yes|no]
--enable-<package>[=yes|no]
```

Values other than yes/no will return an error. --enable-<feature> is synonymous with --enable-<feature>=yes and --disable-<feature> is synonymous with --enable-<feature>=no.

2.5.2 Linking to existing libraries

The user may to link pre-installed libraries with options of the form:

```
--with-<PACKAGE>=PATH
```

where the PATH points to the installation directory typically containing the library and its headers. The PATH argument is mandatory and invalid paths return an error during configuration checks. If a valid library or dependency has been found, then the configuration for the dependency is processed to see if the required headers are available and if a test program that utilizes the library calls can be successfully compiled and linked in order to accumulate the overall LDFLAGS and LIBS to compile SHARP successfully.

2.5.3 Downloading third-party libraries

SHARP allows the user to download and compile third-party libraries through the configure script. Third-party libraries may be downloaded and compiled with options of the form:

```
--download-<PACKAGE>[=yes|no|url]
```

Specifying --download-library[=yes] will download a tarball (.tar.gz file) of the library's source code from a default URL and utilize default, verified workflows to configure, build and install the dependency onto a dependency installation directory. Running "configure –help" provides a list of the default URLs associated with each library. Optionally, the user can also specify --download--library=URL in order to download the library from the given URL and build/install using the same process. Note that the user-provided URL should point to a valid tarball and may not be guaranteed to be compatible with other SHARP libraries and modules.

3 SHARP Drivers

SHARP includes pre-defined drivers to execute both single- and multi-physics simulations with the physics code modules. The advanced user can create their own drivers to solve a multi-physics problem if one of the pre-existing drivers does not fit their purpose (this is advanced usage and not covered in this user manual). The pre-defined drivers with this release of SHARP are provided to run single physics problems with PROTEUS, Nek5000 or Diablo, as well as coupled physics problems (PROTEUS-Nek, PROTEUS-Nek-Diablo). This chapter describes the function and the general usage of these drivers.

3.1 Single Physics Driver

Driver name: single_physics

Description:

This driver is used to execute a single-physics problem (PROTEUS, Nek5000). SIGMA performs only limited operations hands control over to the individual physics code. Proper execution of this driver can be used to verify the calling of all the routines necessary for solving a neutronics, CFD or structural mechanics problem. The driver is generic and thus an argument specifying the physics type must be passed on the command line. This driver can be used with a mesh from the corresponding physics code's native mesh file, e.g. a .rea file for Nek5000, or with a SIGMA formatted (MOAB, .h5m) mesh, because the physics code imports the mesh itself. The results from simulations using this driver should match those from the native executable of the corresponding physics code—e.g., the results of "single_physics -physics_type nek" should match the output of Nek5000 for the same problem—providing one simple means of verification of the operations being performed by the SIGMA interfaces.

Usage:

```
$./single_physics -physics_type proteus -session <problem name>
# Run PROTEUS single physics case
$./single_physics -physics_type nek -session <problem name>
# Run NEK5000 single physics case
$./single_physics -physics_type diablo -session <problem name>
# Run Diablo single physics case
```

3.2 Single Physics Driver with Mesh Import

Driver name: single_physics_with _mesh

Description:

This driver is used to execute a single-physics problem (PROTEUS, Nek5000 or Diablo). It is similar to the "single_physics" driver, except that this driver imports the mesh from an input file written in SIGMA's format (MOAB, .h5m) rather than the native mesh format for the corresponding physics code (e.g. .rea for Nek5000). Results from simulations using this driver should match those from "single_physics" and the native executable of the corresponding physics code.

Usage:

```
$./single_physics_with_mesh -physics_type <proteus or nek or diablo>
-session <problem name>
```

Or

```
$./single_physics_with_mesh -physics_type <proteus or nek or diablo>
<mesh filename>
```

3.3 PROTEUS-Nek Coupled Physics Driver

Driver name: proteusnek

Description:

This driver is used to execute a two physics PROTEUS-Nek5000 problem where iterations between PROTEUS and Nek5000 are controlled by SIGMA. The two codes iterate back and forth on a certain problem until a converged steady-state solution has been reached. The SIGMA framework loads the mesh for both physics codes, maps the solution fields from each physics code to the other mesh, and then transfers the necessary field data. PROTEUS transfers power density information to Nek5000, and Nek5000 transfers temperature and density information to PROTEUS. Currently, the density information is ignored in PROTEUS because PROTEUS must recompute the densities based on its own representation of the materials in the problem.

Usage:

```
$./proteusnek -session <problem name>
```

3.4 PROTEUS-Nek Pseudo Steady State Coupled Physics Driver

Driver name: proteusnek_pseudo_ss

Description:

This is a variation of PROTEUS-Nek5000 driver code. It solves the coupled physics problem to obtain a steady state solution, performs a pre-defined perturbation to the physics, and then recomputes the solution.

Usage:

```
$./proteusnek_pseudo_ss -session <problem name>
```

3.5 PROTEUS-Nek-Diablo Coupled Physics Driver

Driver name: proteusnekdiablo

Description:

This driver is used to execute coupled physics problem with all three physics (PROTEUS-Nek5000-Diablo). The SIGMA framework loads the mesh for the three physics codes, maps the solution fields from each physics code to the other mesh, and then transfers the necessary field data. In this driver, Nek5000 and PROTEUS undergo an inner iteration, passing power and temperature back and forth. Once converged, Nek5000 transmits temperature data to Diablo, which computes mesh displacements and sends these back to PROTEUS and Nek5000. The simulation continues until a desired time specified in the Nek5000 input.

Usage:

```
$./proteusnekdiablo -session <problem name>
```

3.6 Additional Coupled Physics Driver Options

Additional options are available via command line to control any of the coupled physics simulations. These options are listed in **Table** 3.1.

Table 3.1. Command line options for the CouPE Drivers for SHARP

Driver (CouPE) Option	**Option List (Default)**	**Description**
`-coupe_type <string>:`	picard, nk (picard)	Iteration type (one of) picard, nk. The type of global iteration scheme, which can be one of Picard iteration or Newton-Krylov method. For enabling the Newton-Krylov method, the physics wrappers to provide access to the fully consistent, discrete residual vectors and an approximation of the linearized Jacobian operator.
`-coupe_rtol <val>`	Real (1E-06)	Relative tolerance. The relative convergence tolerance for the global coupled nonlinear iteration scheme. This tolerance, along with a constant absolute tolerance (1e-10) controls the success in convergence criteria.
`-coupe_max_it <val>:`	Integer (50)	Maximum iterations. The maximum number of global coupled nonlinear iterations to perform, at every time step in the multiphysics simulation.
`-coupe_ploc_tol <val>:`	Real (5E-06)	Point location tolerance. The minimum tolerance for locating a point on the target mesh, that is used internally when querying an element and computing the inverse mapping. The required accuracy depends on the description and resolution of the geometry and the discrete mesh in each of the physics.
`-proteus_kinetics`	(none)	PROTEUS kinetics flag. If present, perform kinetics in neutronics (requires additional input files). If absent, assume steady state (constant power). *Note that at the time of release (March 2016), this*

		option is in beta testing.
`-physics_type <string>`	proteus, nek, diablo (no default)	Physics type to be performed for single physics drivers only. Specify one of: proteus, nek, diablo.

4 SHARP Verification Tests

The following verification test targets are provided to verify that the single and coupled physics drivers are working correctly. The full set of verification tests can be performed by issuing the following:

```
$cd $SHARP_DIR
$make verify
```

Alternatively, an individual verification test can be performed from the "tests" directory by running make on the appropriate target:

```
$cd $SHARP_DIR/build/tests
$make <target-name>
```

Here, <target-name> is the name of the verification target selected from one of the descriptions below. All of the <target-name> values begin with the string "verify". Each verification test compiles the necessary SHARP drivers, runs a simulation, and compares results to a reference solution.

4.1 Single Physics Verification Tests: Nek5000

Target name: verify_nek_module

Description:

This target runs five single physics Nek5000 verification tests which include both 2D and 3D, laminar and turbulent, with and without conjugate heat transfer problems for SHARP Nek5000 single physics verification. Successful execution of these tests indicates that the Nek5000 module is installed properly.

Usage:

```
$cd $SHARP_DIR/build/test
$make verify_nek_module
```

4.2 Single Physics Verification Tests: PROTEUS

Target name: verify_proteus_module, verify_proteus_kinetics_module

Description:

This target runs several single physics PROTEUS tests (both steady state and kinetics) for SHARP PROTEUS single physics verification. Both serial and parallel tests are performed by default. In order to perform all tests, the user should have available 12 processors. Successful execution of these tests indicates that the PROTEUS module is installed properly.

Usage:

```
$cd $SHARP_DIR/build/test
$make verify_proteus_module
$make verify_proteus_kinetics_module
```

4.3 Single Physics Verification Tests: Diablo

Target name: verify_diablo_module

Description:

The single physics driver for Diablo is not included in this release of SHARP. However, the capability will be released in the near future after a full test to ensure its maturity. Even so, included in the modules/diablo/Tests directory is a small suite of verification problems to test whether the standalone Diablo executable (configured by SHARP) compiles and runs correctly. This test suite does not test all features of Diablo – future releases may include more of the complete Diablo verification suite. The tests which will be covered by the “make verify” command in the Diablo directory are:

- 001 – This is a simple beam bending test using hexahedral elements, with concentrated forces applied to selected nodes at the end, and pressure boundary conditions applied to additional sidesets. This problem is quasi-static (the primary mode in which Diablo is used in the current release of SHARP). This test also ascertains whether MUMPS is operating correctly. The problem runs in parallel via a 4-way decomposition. Finally, mili plot output is tested.
- 001_include – This is a repeat of 001, but it tests whether the “include” capability works with input files.
- 002 – This is a time-dependent version of 002, with some minor differences in problem definition (sidesets versus nodesets for Dirichlet boundary conditions, etc.). It also tests whether HYPRE is hooked up correctly.

4.4 Coupled Physics Verification Tests

Target name: verify_single_physics_nek_sahex, verify_single_physics_with_mesh_nek_sahex, verify_single_physics_proteus_sahex, verify_single_physics_with_mesh_proteus_sahex, verify_proteusnek_sahex, verify_proteusnek_pseudo_ss_sahex, verify_proteusnekdiablo_sahex, verify_sahex

Description:

These targets run tests for verification of all the drivers for the test case "sahex", which is described in the following section. There is a verification target for corresponding to each of the single physics drivers and types, the PROTEUS-Nek drivers, and the PROTEUS-Nek-Diablo drivers. For instance, "verify_single_physics_proteus_sahex" tests "single_physics -physics_type proteus -session sahex", and "verify_proteusnek_sahex" tests "proteusnek -session sahex". The last target, "verify_sahex", is a special target that runs through all of the coupled verification tests for the "sahex" problem, and is equivalent to running the make command on all of the above targets. Successful execution of these tests indicates that the test case "sahex" is compiled and run properly.

Usage:

```
$cd $SHARP_DIR/build/test
$make verify_single_physics_nek_sahex
# verify the single-physics driver with Nek5000
$make verify_single_physics_with_mesh_nek_sahex
# verify the single-physics driver with Nek5000 with mesh
$make verify_single_physics_proteus_sahex
# verify the single-physics driver with PROTEUS
$make verify_single_physics_with_mesh_proteus_sahex
# verify the single-physics driver with PROTEUS with mesh
$make verify_proteusnek_sahex
# verify the coupled-physics driver with Nek5000 and PROTEUS
$make verify_proteusnek_pseudo_ss
# verify the pseudo steady state coupled-physics driver with Nek5000,
PROTEUS.
$make verify_proteusnekdiablo
# verify the coupled-physics driver with Nek5000, PROTEUS, and
Diablo.
$make verify_sahex
# run the above-mentioned tests
```

5 Test Problem: Single Hexagonal Fuel Assembly ("sahex")

The SHARP toolkit includes the full physics input for a test case called "sahex". These example inputs are used in the previous section to verify successful installation of SHARP. They are also useful to understand how to use SHARP for multi-physics calculation. Here we provide step-by-step instructions of how to run the SHARP drivers for the sahex problem. Included are instructions for setup, initialization and visualization of the solution. Reference outputs are also provided such that the user can run the various executables drivers described in the previous section and determine whether the same output is obtained.

5.1 Problem Description

This model consists of a single hexagonal assembly containing six fuel pins with cladding, one central "control" rod with a larger diameter than the fuel pin, sodium coolant and an outer duct wall that encloses all the other components. The geometry of the problem is shown in Figure 5.1. The detailed geometry information and flow condition are listed in Table 5.1 and

Table 5.2 respectively. The specific parameter listed in Table 5.1 is indicated in Figure 5.1.

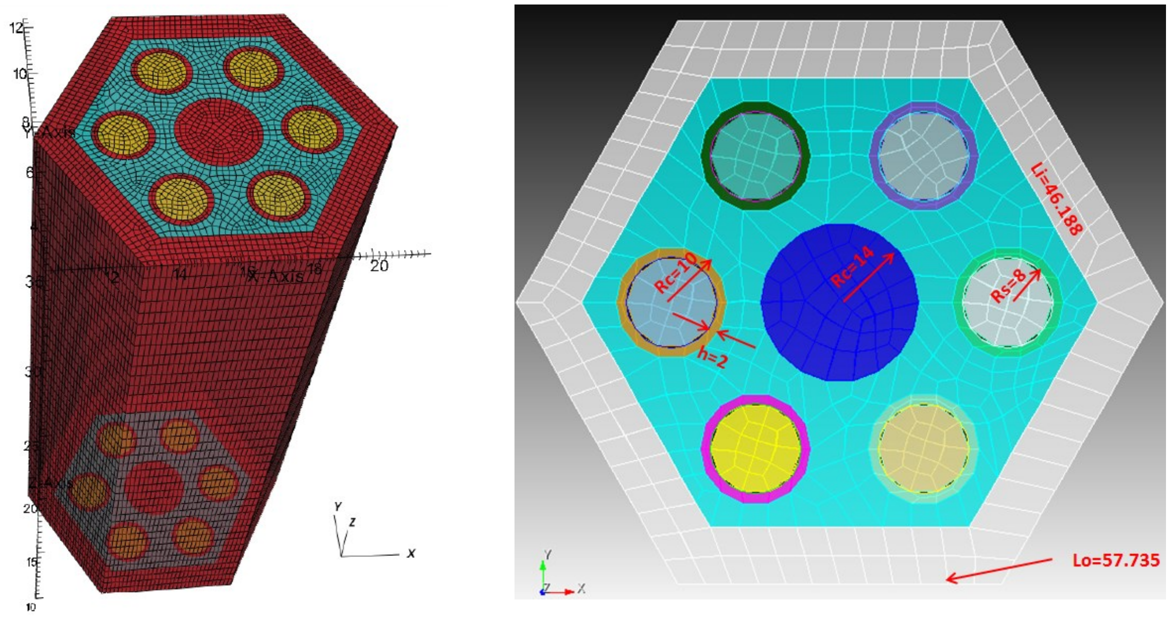


Figure 5.1 sahex problem geometry and mesh.

Table 5.1 Geometry specifications for sahex.

Parameter	Unit (mm)	Parameter	Unit (mm)
Rc	14	*Li*	46.188
Rs	8	*Lo*	57.735
Ra	10	*H*	250
h	2		

Table 5.2 Coolant thermophysical properties and flow conditions for sahex.

Parameter	Value	Remark
ρ	0.85	850 kg/m^3 p1 in *sahex_nek.rea file*
μ	0.26	2.6e-4 Pa·s p2 in *sahex_nek.rea file*
ρC_p	1.08110	C_p=1271.5 J/kg·K P7 in *sahex_nek.rea file*
k	0.7	700K (W/cm·K) P8 in *sahex_nek.rea file*
Re	11700	-
Δt	1e-3	s
V_{in}	65.3896	cm/s

5.2 Input Files

The input files for the sahex problem are listed in Table 5.3. These files include input files for different modules and other control files. The isotopic cross section and delayed neutron input data for the neutronics solver were generated using the MC^2-3 code to obtain parameterized 4- and 9-group cross sections as a function of temperature and density. In the neutronics model, vacuum boundary (non-reentrant) conditions are applied on the top and the bottom surfaces and reflective boundary condition are applied to the sides. In the CFD model, the oulet boundary conditions are applied to the top surface of the fluid domain and adiabatic wall boundaries are applied to the solids. The sides are adiabatic walls.

Table 5.3 sahex Input Files.

PROTEUS	
File Name	***Description***
sahex_proteus.inp	Driver input file, a plain text (ASCII) file, which drives the PROTEUS calculation by specifying solver tolerances, the angular discretization, parallelization options, and other input options. Additionally, the UNIX file paths to the other input files (cross sections, mesh, and material assignment file) are specified here
sahex_proteus.h5m	h5m format mesh file where the block number and sideset number are specified
sahex_proteus_4g.ISOTXS sahex_proteus_9g.ISOTXS	The cross section file that consists of the multigroup cross sections for all isotopes and/or compositions in the problem. The ISOTXS format is the preferred file format for cross section data used by PROTEUS-SN. The MC^2-3 code (Argonne National Laboratory) can be used to process multigroup cross sections in this format. Additionally, the DRAGON code (Ecole Polytechnique de Montreal) [14] has a capability to generate ISOTXS file.
sahex_proteus.anlxs	The cross section file with anlxs file format is a simple ASCII interpretation of the data provided in ISOTXS. Any anlxs file format that is provided is converted into the appropriate ISOTXS file at runtime.
sahex_proteus.assignment	Material assignment file which performs three main functions: (1) define materials or mixtures based on the isotopes in the cross section file, (2) assign these materials to blocks in the mesh, and (3) assign properties (e.g. density, temperature, and material models) to blocks in the mesh.
NEK5000	
sahex_nek.rea	Input file consists of several sections: o Parameters such as viscosity, conductivity, number of steps, time step size, order of the time stepping, frequency of output, iteration tolerances, flow rate, filter strength, etc. o passive scalar and logical switches o Mesh and boundary condition information o Output info like restart conditions
sahex_nek.h5m	h5m format mesh file which can only work under MOAB framework
sahex_nek.usr	The user file where users may specify spatially varying properties (e.g., viscosity), volumetric heating sources, body forces, and so forth. One can also specify arbitrary initial and boundary conditions through the routines useric() and userbc(). The routine userchk() allows the user to interrogate the solution at the end of each time step for diagnostic purposes.

SIZE	The SIZE file that defines the problem size, i.e. spatial points at which the solution is to be evaluated within each element, number of elements per processor etc. The SIZE file governs the memory allocation for most of the arrays in Nek5000, with the exception of those required by the C utilities.
Diablo	
sahex_Diablo.assembly	Input file consists of solver information and material data
sahex_Diablo.subassembly	Input file consists of block ids and boundary conditions
sahex_Diablo.exo	The EXODUS format mesh file and writes the equivalent MOAB (".h5m") file as part of the initialization process
sahex_Diablo.h5m	h5m format mesh file created by Diablo when it initializes
OTHER	
README	Simple description of the sahex problem
Makefile.am	Make file with Driver information
sahex_nek.jou	mesh script for RGG program in MeshKit

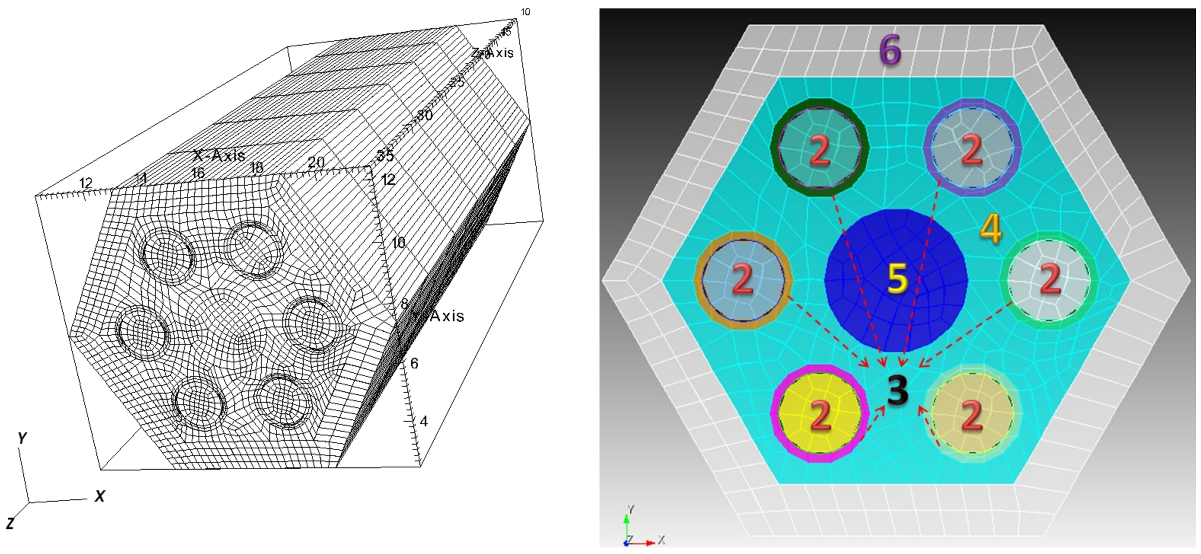


Figure 5.2 NEK5000 mesh and block numbering (sahex_nek.h5m).

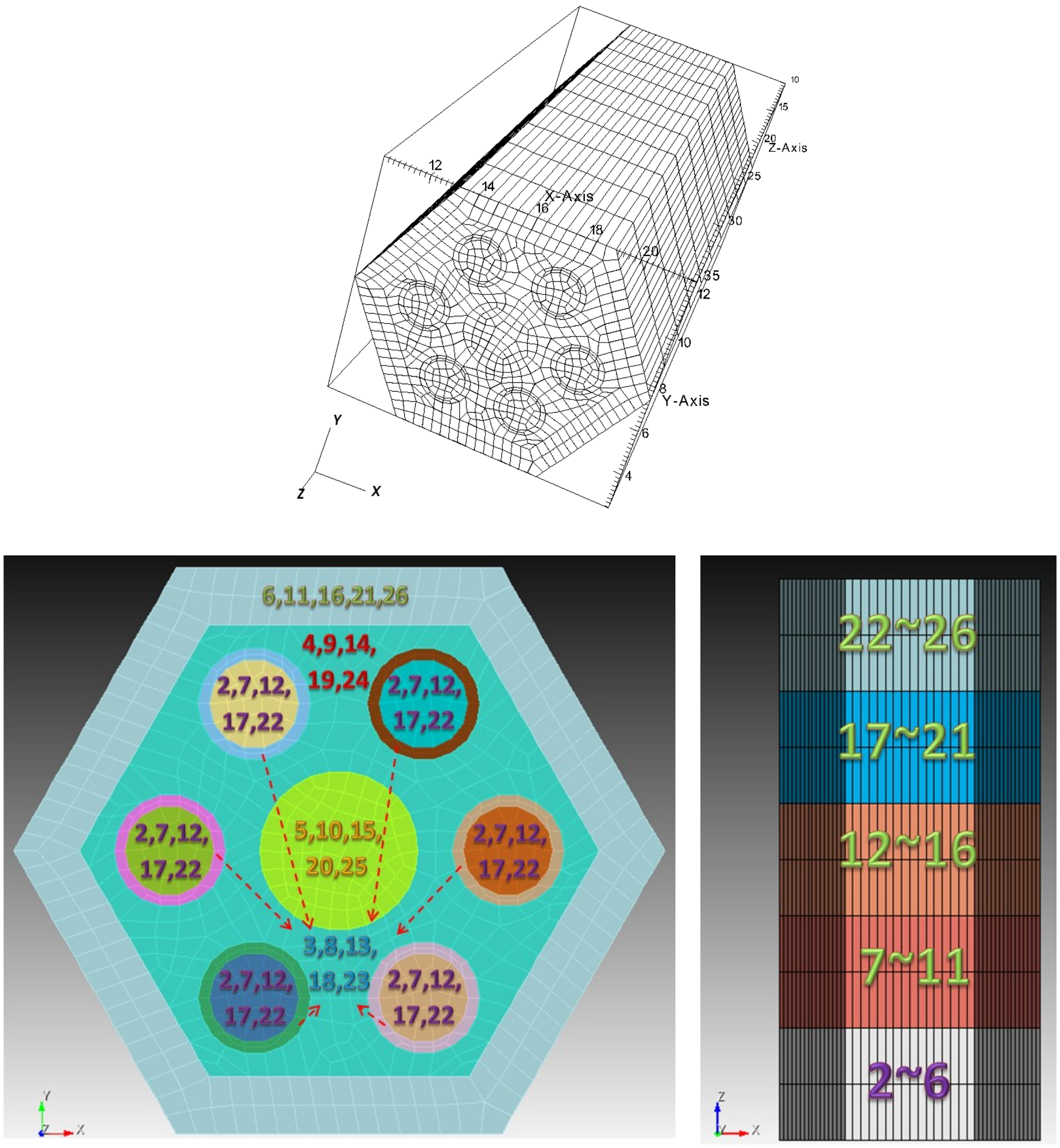


Figure 5.3 Diablo and PROTEUS mesh and block numbering (sahex_proteus.h5m).

The mesh and block number for Nek5000 and PROTEUS are shown in Figure 5.2 and Figure 5.3. For Nek5000, there are 5 blocks for fuel pins; cladding; sodium coolant; control rod and duct wall. For PROTEUS and Diablo, there are 25 blocks which are divided by 5 parts in stream wise direction for different axial height materials. In term of the mesh resolution, the hydraulics solver uses a comparatively coarser mesh than neutronics, consistent with the use of quadratic elements and nature of spectral discretization.

5.3 Running

5.3.1 Steady State Drivers

Step1: Prepare input files

Make sure all the input files are available in the working directory: $SHARP_DIR/build/tests/sahex

Step2: Build drivers

From the working directory, build all the drivers with the following command:

```
$cd $SHARP_DIR/tests/sahex
$make all
```

Diablo related drivers are created if the --enable-diablo option was used to configure SHARP.

Alternatively, the user can build each driver separately with the following command:

```
$make single_physics_sahex
# build single physics driver
$make single_physics_with_mesh_sahex
#build single physics driver to read a mesh file and create an
iMeshInstance
$make proteusnek_sahex
# build PROTEUS-NEK coupled physics driver
$make proteusnek_pseudo_ss_sahex
# build PROTEUS-NEK pseudo-transient coupled physics driver
$make proteusnekdiablo_sahex
# build PROTEUS-NEK-Diablo coupled physics driver
```

The above command lines create executable files single_physics_sahex, single_physics_with_mesh_sahex, proteusnek_sahex, proteusnek_pseudo_ss_sahex proteusnek_sahex and proteusnekdiablo_sahex.

Step3: Run the case

Serial:

```
./single_physics_sahex -physics_type proteus -session sahex
# Run PROTEUS single physics
```

```
./ single_physics_sahex -physics_type nek -session sahex
# Run NEK5000 single physics
./single_physics_sahex -physics_type proteus sahex_proteus.h5m
# Run PROTEUS single physics with iMeshInstance option
./ single_physics_sahex -physics_type nek sahex_nek.h5m
# Run NEK5000 single physics iMeshInstance option
./proteusnek_sahex -session sahex
# Run PROTEUS-Nek5000 pseudo-transient coupled calculation
```

If desired, the user can try out command line option to control aspects of the calculation such as maximum integrations and minimum tolerance (see **Table** 3.1). To run the calculations in parallel with N processors, simply append "mpiexec –n <N>" to the above commands, or the equivalent command for the user's system.

An example script for running the proteusnek_sahex driver on systems that use PBS for job scheduling follows. Save the contents to a file named sahex.pbs.

```
#!/bin/bash
#PBS -N sahex
#PBS -j oe
#PBS -V
#PBS -S /bin/bash
#PBS -m bae
cd $PBS_O_WORKDIR
NHOSTS=`cat ${PBS_NODEFILE} | wc -l`
make proteusnek_sahex
mpiexec –n $NHOSTS ./proteusnek_sahex -session sahex
```

This PBS script is rather generic. It assumes that the "qsub" command is issued from the sahex directory. Also, the computing resources are not specified, and so the user must do so on the command line. Specifying the computing resources is platform-specific, as you need to know the number of processes to use on each node. For instance, on the ANL cluster Blues, there are 16 cores per node. For this machine, you could submit the job to run on 64 processes using the following commands:

```
$cd $SHARP_DIR/build/tests/sahex/
$qsub –lnodes=4:ppn=16 sahex.pbs
```

Step 4: Check the output files

Generally in addition to the text output, the following output files are created after calculation:

PROTEUS: sahex_4g_L1T2.hdf5.xx

NEK5000: sahex_nek.fldxx

MOAB: sahex0001~ sahex0xxx.h5m

Where hdf5 files include the neutronic information and fld files include the thermal hydraulic information while h5m file include both.

5.3.2 Pseudo-transient

The type of transient examined in the paper is a simplified loss-of-heat-sink, where the temperature of the fluid at the inlet boundary is specified as a function of time and the evolution in the coupled fields is computed. This simulates an accident scenario when the heat exchangers fail to remove excess heat from the coolant, thereby increasing the inlet temperatures steadily, causing feedback effects from different sources to interact nonlinearly between the physics.

$$T(t) = \begin{cases} T_0, & t \leq t_0 \\ T_0 + \alpha T_0 tahh\left(\frac{t - t_0}{\Delta t}\right) & t \geq t_0 \end{cases}$$

where T_0 is the transient initiation time, T_0 is the initial converged temperature solution, Δt is the duration of the transient at the inlet and α is the damping parameter to control the magnitude of the perturbation (typically 0.2). User can run the case with the following command:

```
./proteusnek_pseudo_ss_sahex -session sahex # Run PROTEUS-Nek5000
pseudo transient coupled calculation
```

5.3.3 Analyzing

5.3.3.1 Steady State

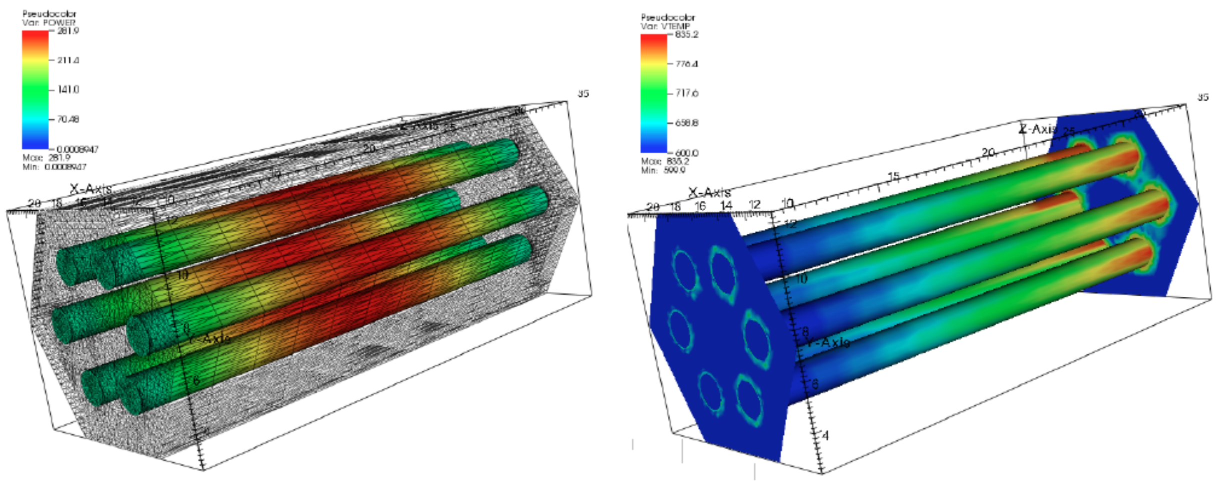


Figure 5.4 Power profile and temperature distribution for steady state solution

The profile of the integral power based on the angular flux computed from solving the Boltzmann neutron transport equation and the temperature profile from thermal-hydraulic solver as shown in Figure 5.4 . The decoupled profiles are physically meaningful and provide a good initial of the power distribution shifts towards the inlet of the core due to lower material density at the top of the assembly, while the peak temperatures are observable near the outlet since the coolant temperature is monotonically increasing.

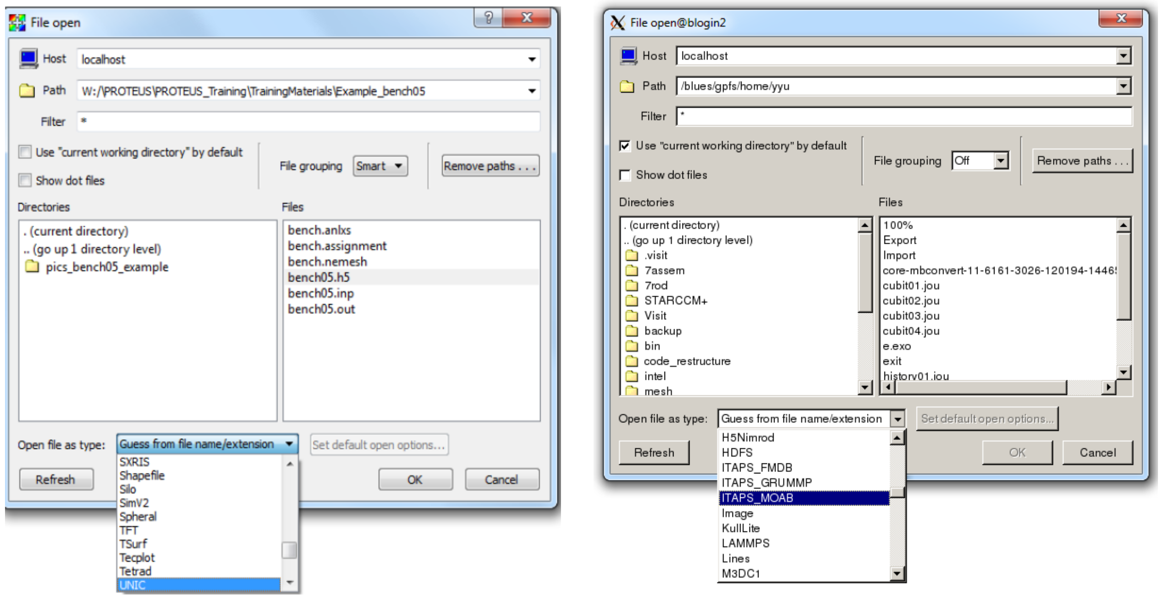
```
Visnek sahex_nek
```

to crease sahex_nek.5000 file which can be loaded into VisIt to visualize the results or use vis.nek3d script

Script of vis.nek3d:

```
NEK5000
version: 1.0
filetemplate:  sahex_nek.fld%02d
firsttimestep: 1
numtimesteps: #
Remember to edit numtimesteps: # (the last time step)
```

For PROTEUS, choose UNIC for open file as type as shown in Figure 5.5 to load .hdf5xx file. Alternatively, we can load h5m file to visualize the result by using ITAPS_MOAB type.



File open

Host: localhost

Path: W:/\PROTEUS\PROTEUS_Training\TrainingMaterials\Example_bench05

Filter: *

File grouping: Smart

Open file as type: Guess from file name/extension

Directories: . (current directory), .. (go up 1 directory level), pics_bench05_example

Files: bench.anlxs, bench.assignment, bench.nemesh, bench05.h5, bench05.inp, bench05.out

File open@blogin2

Host: localhost

Path: /blues/gpfs/home/yyu

Filter: *

Use "current working directory" by default: ✓

File grouping: Off

Open file as type: Guess from file name/extension

Directories: . (current directory), .. (go up 1 directory level), .visit, 7assem, 7rod, STARCCM+, Visit, backup, bin, code_restructure, intel, mesh

Files: 100%, Export, Import, core-mbconvert-11-6161-3026-120194-1446!, cubit01.jou, cubit02.jou, cubit03.jou, cubit04.jou, e.exo, exit, history01.jou

Figure 5.5 Load .hdf5 and .h5m file in VisIt

The power distribution generated by adding a pseudocolor plot of the Power variable from sahex_4g_L1T2.hdf5.xx is shown in Figure 5.6 and the temperature distribution on different elevations from sahex_nek.fldxx is shown in Figure 5.7. Some work flow in VisIt is shown in Figure 5.8. Several successive refinements of the neutronics and hydraulics meshes are performed. The convergence result of k_{eff} based on a reference mesh solution is listed in Table 5.4. User can find the refined mesh in ./refinement directory. Users should be able to perform more elaborated mesh convergence study by generating their own meshes without doubt.

Table 5.4 Mesh convergence study on k_{eff}

Element #	Element size	time	keff	Error(%)
7590	0.7144	44.64	0.67387473	0.42
54740	0.3573	63.39	0.67671801	0.114
437920	0.1791	84.27	0.67751902	0.0315
3503380	0.0895	162.93	0.67775429	reference

It is also imperative to note that the parallel performance of the solvers (on 32 processors) measured using the computational cost per Picard iteration increases nearly linearly with the number of degrees-of-freedom.

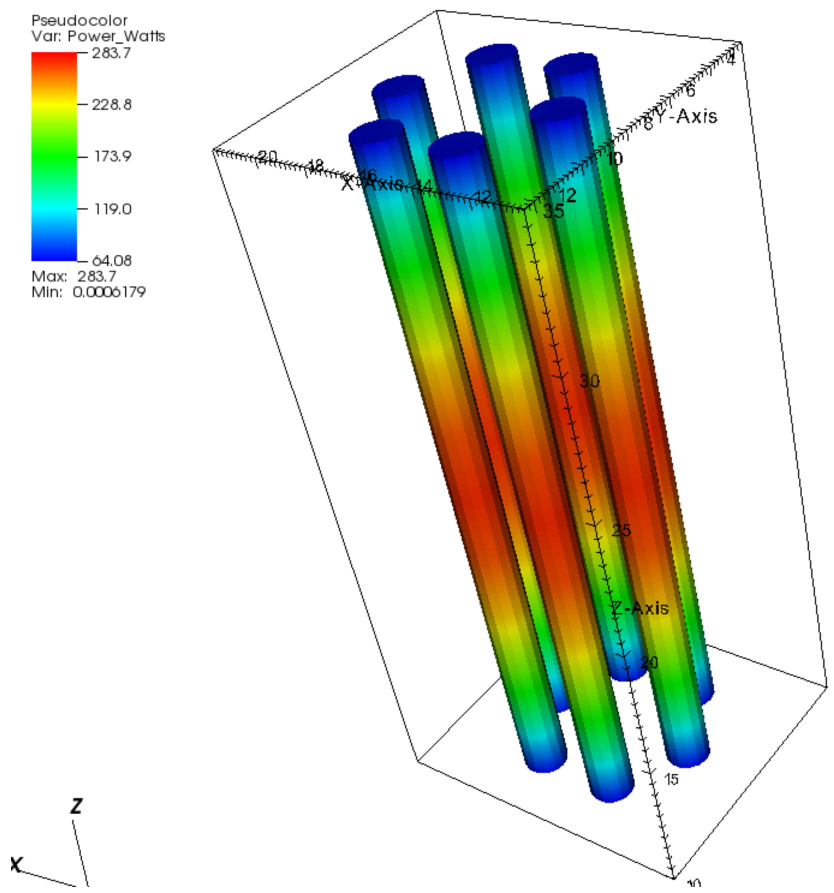
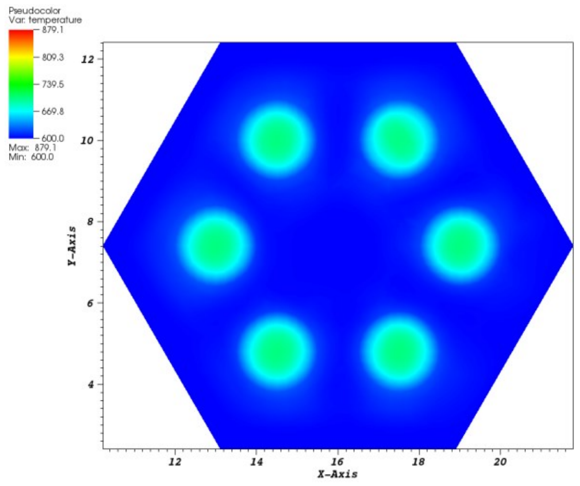
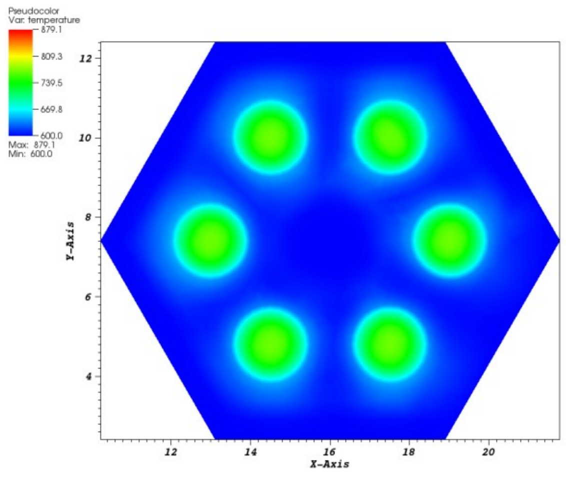


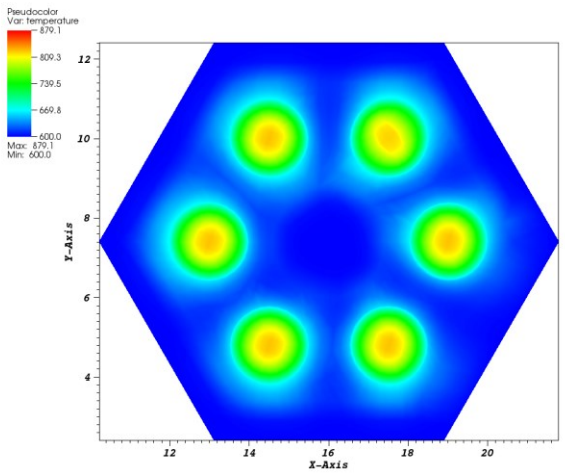
Figure 5.6 Power_Watts distribution.

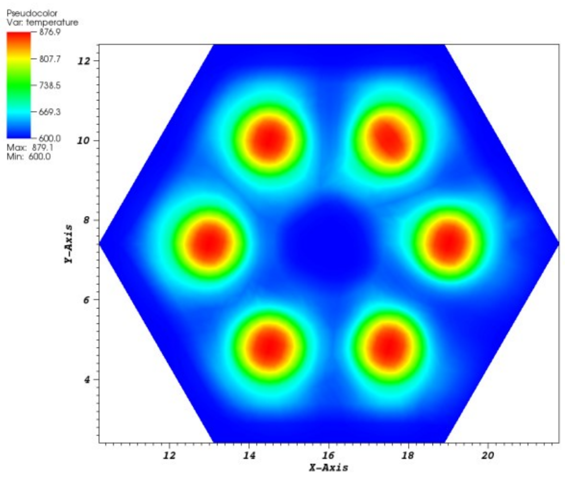


1/4 Span



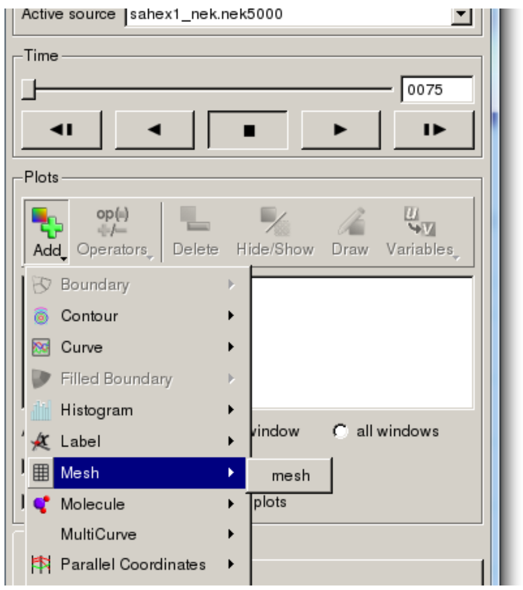
1/2 Span

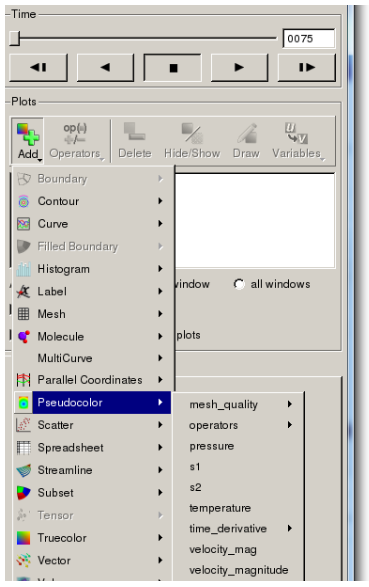




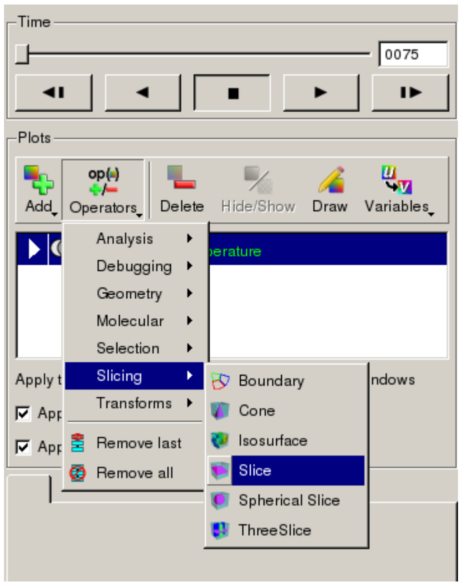
3/4 Span 1 Span

Figure 5.7 Temperature distributions on different elevations.

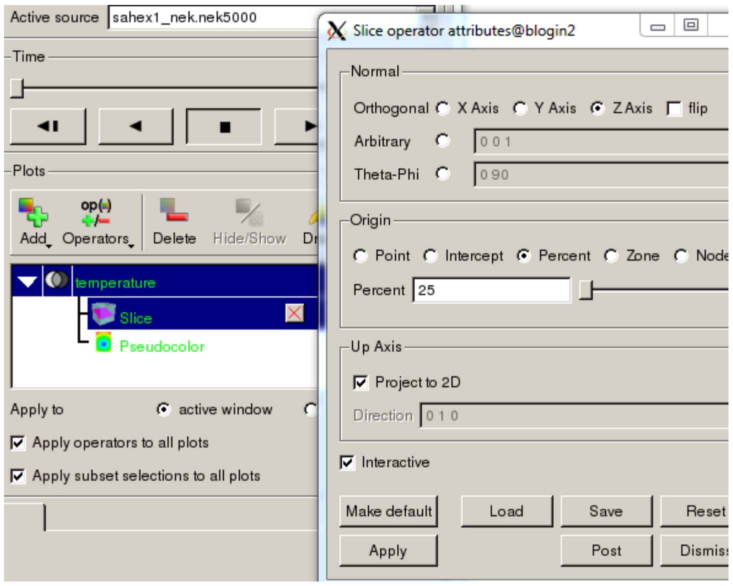




Visualize mesh Create contour for variables



Create slice for contour of variables



Apply to [✓ active window]
☑ Apply operators to all plots
☑ Apply subset selections to all plots

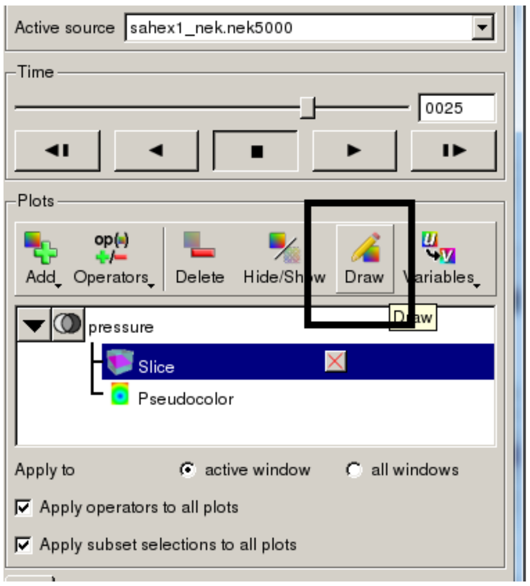
Normal
Orthogonal ○ X Axis ○ Y Axis [✓ Z Axis] ☐ flip
Arbitrary ○ 0 0 1
Theta-Phi ○ 0 90

Origin
○ Point ○ Intercept [✓ Percent] ○ Zone ○ Node
Percent 25

Up Axis
☑ Project to 2D
Direction 0 1 0

☑ Interactive

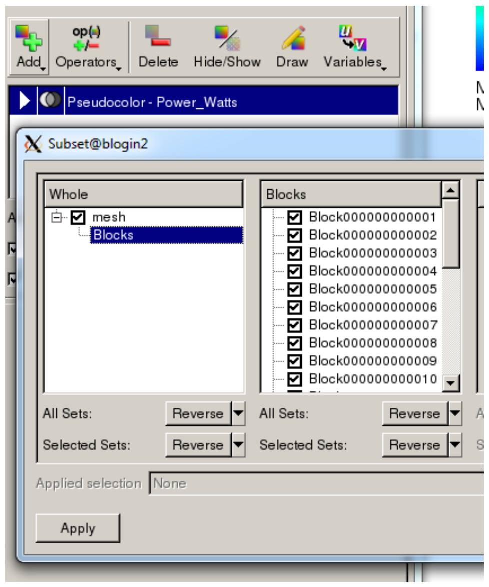
Adjust slice location



Active source sahex1_nek.nek5000

Apply to [✓ active window] ○ all windows
☑ Apply operators to all plots
☑ Apply subset selections to all plots

Draw the plot



Blocks
☑ Block000000000001
☑ Block000000000002
☑ Block000000000003
☑ Block000000000004
☑ Block000000000005
☑ Block000000000006
☑ Block000000000007
☑ Block000000000008
☑ Block000000000009
☑ Block000000000010

Applied selection None

Select blocks to be visible

Figure 5.8 VisIt work flow.

5.3.4 *Pseudo-transient*

Once the Initial Conditions are converged, the loss-of-heat-sink simulation is initiated at t=t_0, by updating the inlet boundary conditions to increase inlet boundary temperature from 600 to 720K during the transient. The total power in the assembly is specified as user input and power distributions are normalized accordingly.

```
#include "../src/physics/impls/proteus/proteusimpl.h"
PetscErrorCode PerformPerturbation_PROTEUS(Physics proteus)
{
  PetscReal perturbation_factor = 2.0;
  // scale the total power by 2.0; should see
  // increase in temperature and decrease in density
  SN2ND_ScaleTotalPower(&perturbation_factor);
  PetscFunctionReturn(0);
}

#include "../src/physics/impls/nek/nekimpl.h"
PetscErrorCode PerformPerturbation_NEK(Physics nek)
{
  PetscReal perturbation_factor = 1.2;
  // scale the input boundary condition power by 1.2; should see
  // increase in temperature and decrease in density
  NEK_Perturb_BCVars(&perturbation_factor);
  PetscFunctionReturn(0);
}
```

In all the cases, the number of sub-cycling steps performed in thermal-hydraulics was specified to resolve the transient change in temperature. Several transients have been performed to test for sensitivity of the coupled field solutions to different feedback effects. Figure 5.9 shows the change in k_{eff} as a function of normalized time for different types of coupled feedback effects optionally turned on. As the frequency of coupling is increased, the accuracy of the coupled physics solution improves since the computed criticality converges towards the reference. The flow time of the sodium through the assembly is 0.9 s (characteristic time scale), and the overall time-steps are reduced consistently to resolve this spatial and temporal scales in successive simulations starting with Δt_{coarse}=0.02s. Note that feedback based on both temperature and density are necessary to show the complex nonlinear coupling between the neutronics and thermal-hydraulics physics for this test problem since the case where only Doppler feedback is considered shows larger sensitivity to the inlet temperature change. In other words, the density and Doppler expansion feedback are competing effects as validated from theory and experimental observations. The total power decrease in the assembly as the transient progresses can be observed in Figure 5.10a. The corresponding evolution of the temperature profiles is shown in Figure 5.10b. Note that the significant change in power profile corresponding to only a minor penetration of the high temperature front within the domain indicates a very fast response (high sensitivity) to the boundary condition in the system.

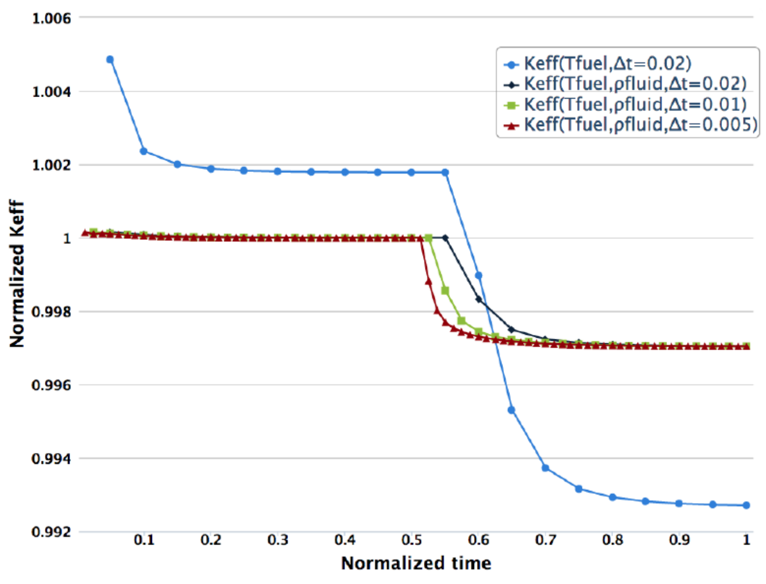


Figure 5.9 k_{eff} transient profile as a function of feedback and temporal resolution

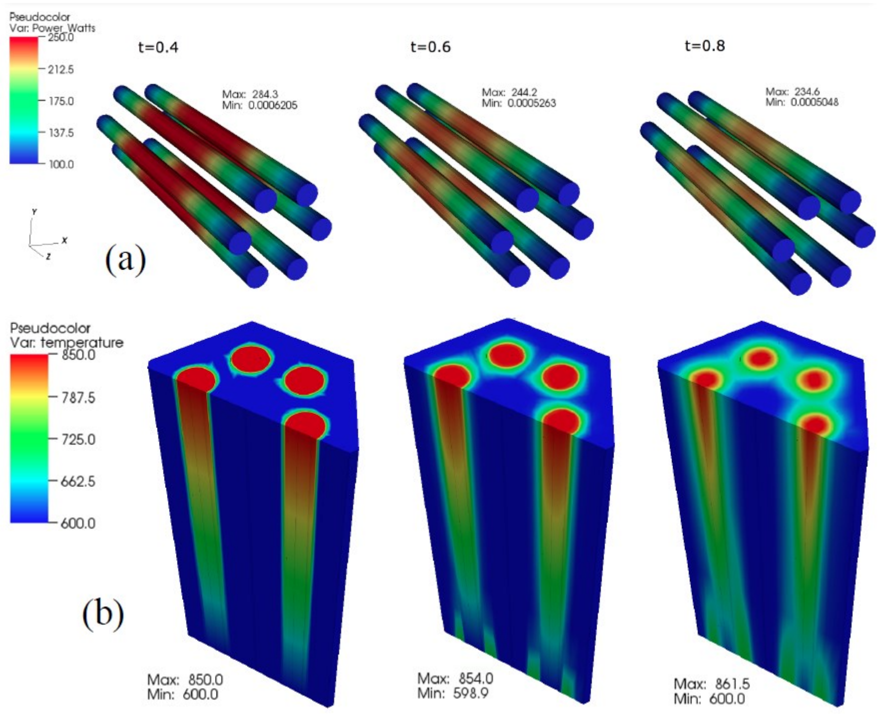


Figure 5.10 Transient evolution of coupled field profiles at the beginning, during and at the end of the perturbation a) power distribution (W) b) temperature (K)

The coupled physics simulation capability with SHARP framework was tested on the sahex problem for a loss-of-heat-sink transient and the results obtained have been verified by spatial and temporal solution convergence studies. The sensitivity tests lead to important conclusions on:

i) The time-step size necessary for this transient to maintain accuracy.

ii) The importance of the inclusion of all types of feedback effects.

6 Create a New Test

Once the user is familiar with running the "sahex" example problem, they can proceed with creating their own test case. This section describes how to create a new test case both from a configuration/installation standpoint and also from an input file standpoint. The user should first duplicate the contents of the sahex directory in a new directory called "mytest" (or some other new name). Then, the input files in the new directory should renamed "mytest" (in our example) instead of "sahex" and filled in with the content required to model "mytest". Certainly, users can used their own input files to create their own case by following this procedure.

6.1 Workflow to Configure/Compile a New Test

Step 1: Copy the sahex example into a new directory in tests/ and rename the input files to follow the expected convention. SHARP requires that input files follow a particular naming convention in order to automatically link the driver to the appropriate input files.

```
$cd ~/SHARP/tests
$cp -r sahex mytest
```

Change name of all the input files from sahex to mytest. For instance, sahex.rea becomes mytest_nek.rea; sahex_proteus.inp to mytest_proteus.inp. Remember to change content in mytest_proteus.inp and kinetics.inp. For example,

```
SOURCEFILE_MESH          mytest_proteus.h5m
SOURCEFILE_XS            mytest_proteus_4g.ISOTXS
SOURCEFILE_MATERIAL      mytest_proteus.assignment
EXPORT_FILE              mytest_4g_L1T2.hdf5
```

Step 2: Modify the makefile.am and configure.ac files to include information about the new problem "mytest". Modify $SHARP_DIR/tests/Makefile.am by including mytest in new SUBDIRS:

```
SUBDIRS = dbgprb sahex sahex_core xx09 xx09_core mytest
```

Modify /tests/mytest/Makefile.am by changing the PROBLEM variable and driver name:

```
PROBLEM = mytest

# Drivers
single_physics_mytest_SOURCES = ../drivers/single_physics.cxx
single_physics_with_mesh_mytest_SOURCES = ../drivers/single_physics_with_mesh.cxx
```

```
if SHARP_ENABLE_PROTEUS
proteusnek_mytest_SOURCES = ../drivers/proteusnek.cxx
proteusnek_pseudo_ss_mytest_SOURCES = ../drivers/proteusnek_pseudo_ss.cxx

if SHARP_ENABLE_Diablo
DRIVERS += proteusnekdiablo_mytest
proteusnekdiablo_mytest_SOURCES = ../drivers/proteusnekdiablo.cxx
endif
```

Modify $SHARP_DIR/configure.ac to add the new test directory:

```
# Create test directories
SHARP_PREP_TESTDIR([mytest])
```

Step 3: Rebuild SHARP

```
$ cd $SHARP_DIR
$ ./bootstrap
$ cd build
$ make
```

Make the drivers.

```
$ cd $SHARP_DIR/build/tests/mytest
$ make all #create all drivers
$ make single_physics_mytest # create specific driver
```

After this procedure, the user has created a new template problem which can be successfully configured with SHARP. The individual physics code input files can now be updated with the appropriate input content.

6.2 Input File Preparation for SHARP

6.2.1 Nek5000 Input Files for SHARP

In order to employ Nek5000 module, the user must define the simulation by preparing geometry and parameters file (.rea) case set-up file (.usr), problem size file (SIZE) and mesh file (.h5m). Since SHARP has some special requirements on these files, it is recommended that the user should copy the pre-existing SAHEX files (except mesh file) from the sahex directory as a template. The user can customize these files in accordance with their specific problem. The Nek5000 user manual link provided in chapter 1 contains detailed instructions on modifying these files. The user needs to create the mesh and convert it to .h5m format file with mbconvert installed in MOAB library with the following command:

```
mbconvert  <target mesh file(.exo or .h5m)> <output mesh file(.h5m)>
```

It’s worth noting that Nek5000 only accept Hex27 elements. The user needs to convert conventional Hex8 elements to Hex27 elements with Cubit or other three party tools before using mbconvert. In the same directory, user can use mbpart to partition the mesh:

```
mbconvert <processor number> <original mesh file(.h5m)> <output mesh
file(.h5m)>
```

After generating the mesh file, user need to define the mesh information such as block ID, material number and boundary conditions in mesh section of .rea file. The setup in rea file for a simple example of conjugate heat transfer is shown in Figure 6.1. The content of the following parts need to be filled with caution, otherwise, Nek5000 shall fail to read the mesh.

Part1: File name of h5m file.

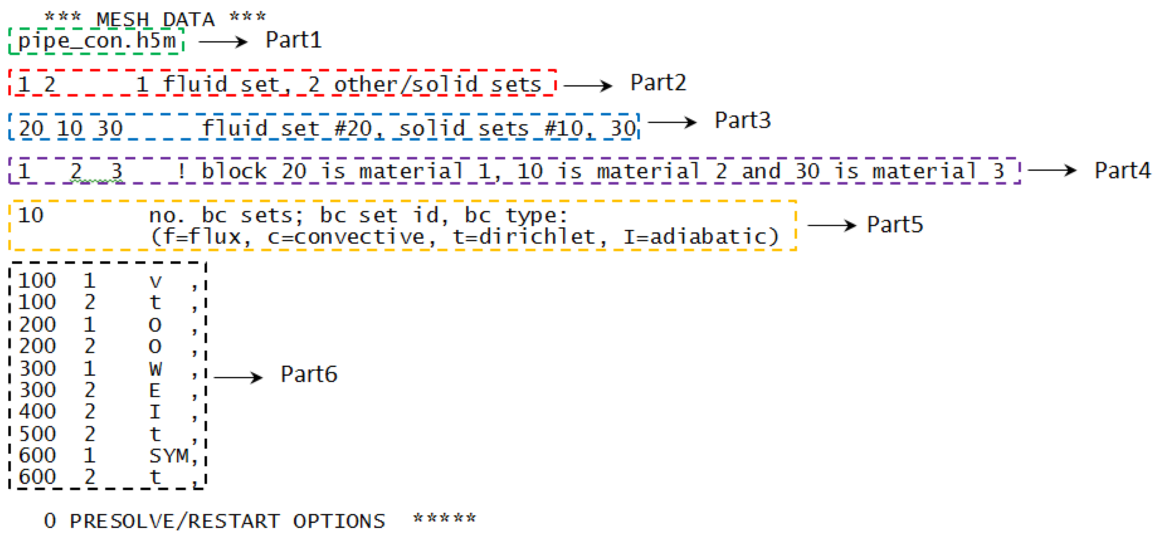
Part2: Number of fluid regions and number of solid regions.

Part3: Fluid blocks ids and solid block ids.

Part4: Material number, the sequence of this number should be in line with the sequence of the blocks ids in part 3.

Part5: Number of boundaries.

Part6: Definition of boundary conditions. First column: sidesets number. Second column: region ID, 1for fluid and 2 for solid. Third column: boundary conditions.



```
  *** MESH DATA ***
pipe_con.h5m
1 2       1 fluid set, 2 other/solid sets
20 10 30       fluid set #20, solid sets #10, 30
1   2  3     ! block 20 is material 1, 10 is material 2 and 30 is material 3
10          no. bc sets; bc set id, bc type:
            (f=flux, c=convective, t=dirichlet, I=adiabatic)
100  1     v  ,
100  2     t  ,
200  1     O  ,
200  2     O  ,
300  1     W  ,
300  2     E  ,
400  2     I  ,
500  2     t  ,
600  1     SYM,
600  2     t  ,
  0 PRESOLVE/RESTART OPTIONS  *****
```

Figure 6.1 Mesh section in rea file

6.2.2 PROTEUS Input Files for SHARP

PROTEUS defines the simulation by driver input file (.inp), cross section file (.ISOTXS), material assignment file (.assignment) and mesh file (.h5m). When performing multi-physics neutronics problems with the PROTEUS module, some additional input options are required and/or optional which do not apply in standalone mode.

- **Required: Mesh in h5m format**

PROTEUS requires the mesh for a coupled problem to be in h5m format. Only exterior sidesets can exist in the mesh. Any interior sidesets must be removed for PROTEUS. Note that PROTEUS assigns unique materials on a block-wise basis – therefore temperature feedback received from Nek5000 is averaged over the entire block on element volume-weighted basis. If a finer distribution of temperatures is required for feedback, the user must refine the block size in the mesh. PROTEUS communicates powers on an element-wise basis.

- **Required: Cross Section Data at Two Temperatures**

PROTEUS requires cross sections at two temperatures in order to perform temperature feedback (interpolate cross sections at specific temperatures during the simulation). The ISOTXS (cross section file) must contain two sets of data for each isotope, one at a lower temperature, and one at a higher temperature. Since a particular isotope name can only be associated with data at one temperature, the following naming convention must be used. The isotopes with lower temperature data must have names fewer than or equal to 6 characters. The isotopes with higher temperature data are named identically to the lower temperature data name, with underscores and a capital T appended to the name in order to make the name 8 characters.

Example: U238 (lower temperature data for U-238) and U238_T (higher temperature data for U-238)

- **Required: Material Assignment File Properties**

For coupled problems, the density of each region (block) must be given as Density(g/cc). The concentration of each isotope in the mixture must be given in atom % rather than weight %.

Initial temperatures per block must be given, consistent with Nek5000, using the keyword TEMPERATURE(K). Material models are optional.

- **Required: Specification of Kinetics Solve via Command Line**

To perform a kinetics solve rather than a steady state solve, the kinetics input file must be named kinetics.inp and the option "-proteus_kinetics" must be passed to the driver.

- **Required: Specification of Timesteps in Kinetics Input**

To perform kinetics solve in a multi-physics coupled problem, only one timestep should be specified in the PROTEUS kinetics input. The assignment file at this timestep must be identical to the initial condition assignment file. The time at the end of the timestep will be overridden by the Nek5000 timestep size. The intervals per timestep will be used to define spacing, but if the USE_RADAU option is turned on, the intervals per timestep should not matter.

- **Optional: Density Computation as a Function of Temperature**

The SN2ND_Computes_Density keyword in the PROTEUS control input file may optionally be defined as YES to allow PROTEUS to update isotopic densities based on material temperature models. If this option is defined as YES, then the MATERIALMODEL property must be assigned to all regions in the material assignment file. Currently, only MATERIALMODEL 0 and MATERIALMODEL 1 are supported. MATERIALMODEL 0 indicates that no density should be recomputed for this block based on temperature. MATERIALMODEL 1 indicates that density should be recomputed for any sodium isotopes in the block based on temperature. If the SN2ND_Computes_Density keyword is undefined or NO, PROTEUS will not update any material densities based on temperature. To be clear, this keyword option only affects recalculation of density based on temperature (not mesh movement).

- **Automatic Density Computation as a Function of Mesh Deformation**

The user should understand that if mesh deformation occurs during the simulation, PROTEUS will automatically re-compute the densities of each isotope to *conserve mass*. The SN2ND_Computes_Density and MATERIALMODEL properties will supersede this density calculation if provided. For example, sodium densities will be computed according to temperature, whereas fuel densities will be calculated according to mesh deformation.

6.2.3 Diablo Input Files for SHARP

Diablo is a Multiphysics implicit finite element code with an emphasis on coupled structural/thermal analysis. In the SHARP framework, it is used as the structural solver, and may also be used as the mesh smoother.

In the current SHARP implementation, Diablo receives temperatures from Nek5000, which are calculated in a coupled fashion with the PROTEUS neutronics code. The change in temperatures induces thermal stresses in the structural model. Diablo then performs one or more Newton (or Quasi-Newton) iterations until the structural solve is converged (meaning the structure is once more in equilibrium), and then returns the deformed configuration to the rest of SHARP.

Diablo may also be used as the mesh smoother for SHARP (currently this is the only method tested). Diablo accomplishes this task by solving a solid mechanics problem for a domain that includes the entire domain, including the fluid domain. The two tools by which this is accomplished are “pseudo-materials” and “duplication”.

Pseudo-materials are simply choices of material parameters for regions which are not structural, e.g. the fluid regions and other regions for which a structural calculation is not desired. Typically one would choose an elastic material (e.g. material model 1, 15, or 27) with small values of elastic constants, e.g. 1/100th or 1/1000th of the typical values of the structural materials. The smaller the value the more accurate the structural solution, but the more difficult the conditioning of the resulting linear system is, which may result in a less efficient solution. Choosing a direct linear solver such as WSMP or MUMPS is generally preferable because they are less sensitive to the condition number of the matrix. Typically it is advantageous to provide pseudo-materials with a realistic value of the coefficient of thermal expansion.

Duplication allows an element set to be “duplicated”, including all the associated nodes and elements. This allows the mesh smoother to operate independently, in some sense, from the solid mechanics solution.

An example of duplication is the provided by the following diagram:

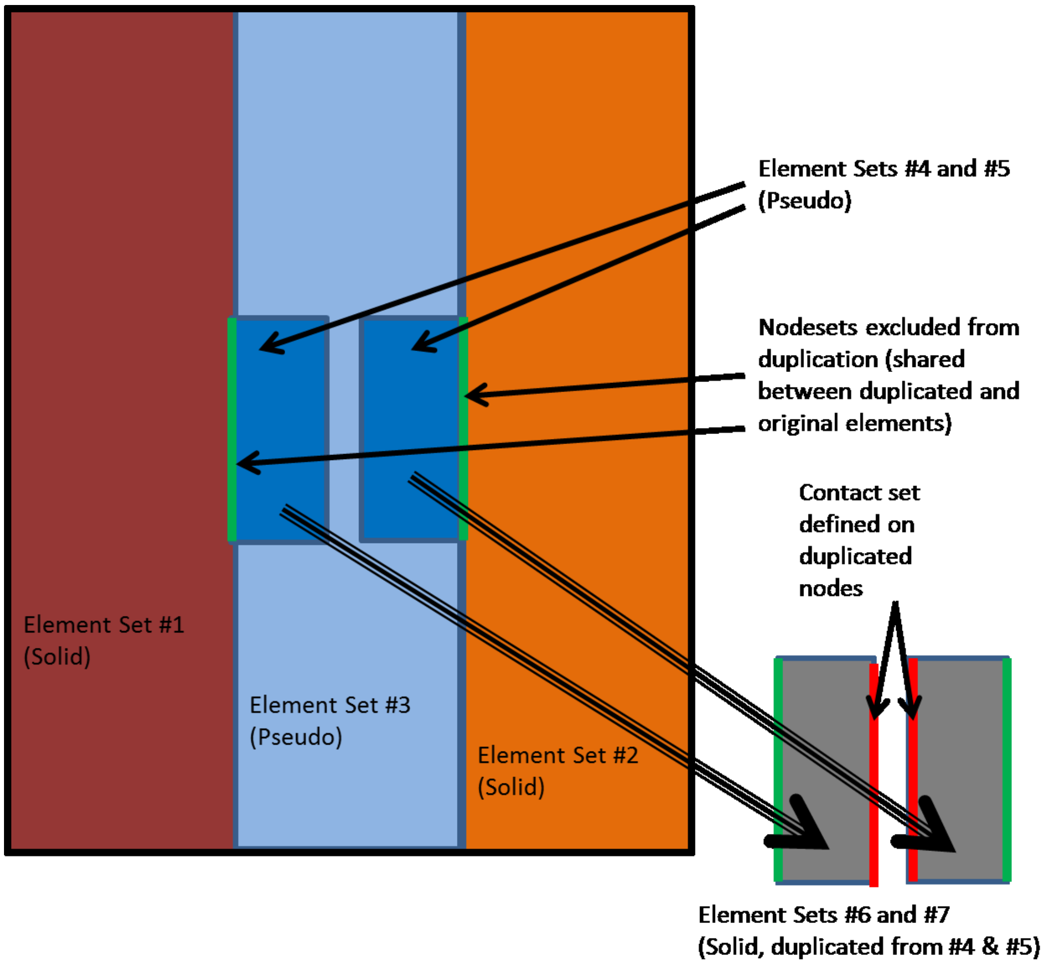

Element Sets #4 and #5
(Pseudo)
Nodesets excluded from duplication (shared between duplicated and original elements)
Contact set defined on duplicated nodes
Element Set #1
(Solid)
Element Set #3
(Pseudo)
Element Set #2
(Solid)
Element Sets #6 and #7
(Solid, duplicated from #4 & #5)

Figure 6.2 SHARP/Diablo usage

In the figure, element sets #1 and #2 represent structural material that is shared between the rest of SHARP (e.g. PROTEUS and NEK) and Diablo. Element Set #3 represents fluid material that is modeled as pseudo-material in Diablo. Element sets #5 and #6 represent material that is structural, but represented as fluid in Nek and PROTEUS (e.g. pads). In order to represent the structural response, the element sets are duplicated as element sets #6 and #7. A contact set between the two element sets is defined on the duplicated nodes. The green node set (or, equivalently, sideset) represents nodes that are shared by both the duplicated and the unduplicated elements (in order that the duplicated elements may be coupled structurally to the rest of the mesh). By means of selective use of duplicated element sets, and duplicated and excluded element sets, the structural response of a reactor may be modeled, as per Figure 2.

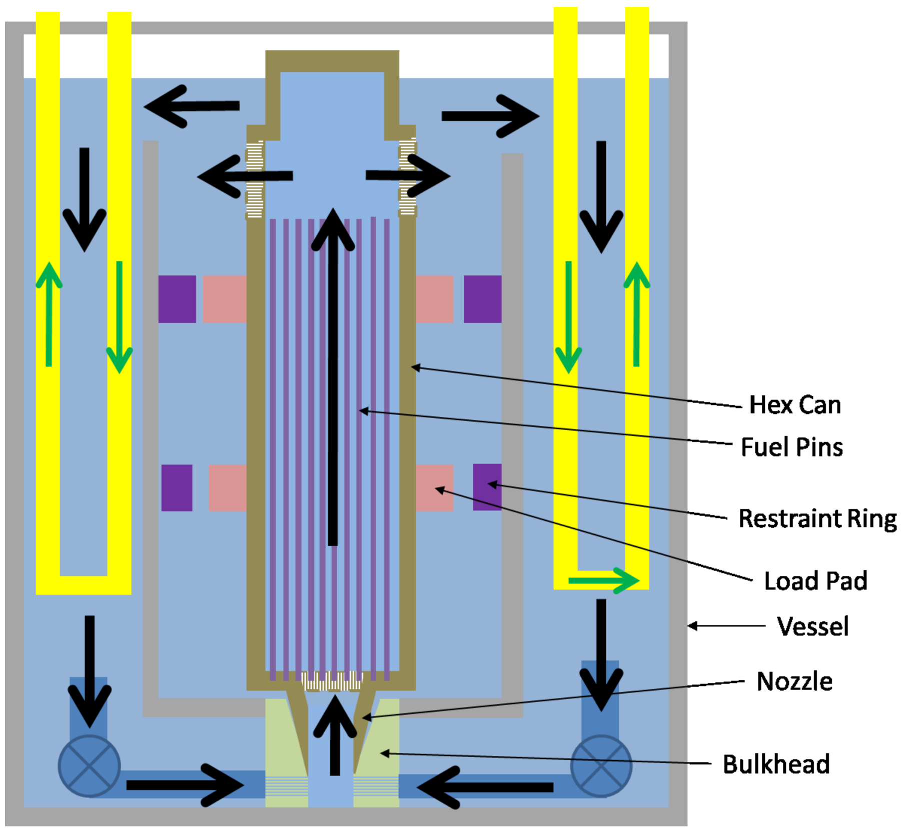


Figure 6.3 Notional reactor model

All valid Diablo syntax is valid within the SHARP framework, with the exception that time stepping (e.g. choice of time step) is controlled via SHARP, and all temperature information is also provided by SHARP.

Duplication and the associated other features are provided only within the context of the exodus reader. The Diablo manual consistent with the current release of SHARP is in the modules/Diablo/documents directory.

To create a duplicate element set, use the following command or variations thereof in the subassembly file within an element set definition.

```
#start_element_topo_exodus
#exodus_block_id 5
#duplicate_nodes_flag .TRUE.
#exclude_duplicates_sideset 5
#exclude_duplicates_nodeset 6
```

```
#end_element_topo_exodus
```

In a Neumann, Dirichlet, or Contact Set definition, the duplicated nodes can be selected (instead of the original nodes) via the #map_nodes_flag.

```
#start_bc_topo_exodus
#exodus_sideset_id 5
#map_nodes_flag .TRUE.
#end_bc_topo_exodus
```

7 Frequently Asked Questions

This chapter describes the frequently asked questions when using SHARP.

1. How do I fix the error "checking whether the C compiler works... no" when configuring SHARP?

Answer: This error could appear in various conditions. It is most often caused by not using the proper version of MPI or pointing to the wrong MPI directory. Users can check chapter 2 to find the proper version of MPI and use the following command to specify the MPI directory:

```
export MPI_DIR=<MPI directory>
```

2. How do I fix the error that says "additional relocation overflows omitted from the output" when compiling the drivers?

Answer: This error is usually caused by the setup which doesn't meet the per-processor memory requirements for Nek5000. This problem can be fixed by reducing "lelt" in the Nek5000 SIZE file.

3. How do I fix the error that says "Missing temperature or mass density in ImportTandD Missing temperature or mass density" when running the calculation?

Answer: The error shows when temperature of regions is not specified in PROTEUS material assignment file (for instance sahex_proteus.assignment) with the following format:

```
REGION_PROPERTY REGION_000000002   TEMPERATURE(K)  800
```

4. How do I fix the error "Density/Temperature varies by more than 15% between the last iteration and the current one" when running the calculation?

Answer: This error indicates that the initial conditions of density or temperature for PROTEUS and Nek5000 are not consistent and the difference is more than 15%. In the current version of SHARP, the error can only be fixed by checking the consistency of the initial conditions for the two modules.

5. When running the calculation, how to fix the error that says :

```
read .rea file
ABORT: nelv is invalid in nekmoab_proc_map
nelv, lelv =            0        6496
call exitt: dying ...
```

Answer: This error is due to an input set up mistake in rea file for Nek5000. User should be able to fix that issue by checking if the block ID in the mesh is consistent with the number specified in rea file. The detailed setting is discussed in chapter 5.1. User ought to be able to get the information of the mesh of h5m format by using the following command gives

mbsize –m <mesh file name>

Generally speaking, mbsize locates at /build/libraries/moab/moab-4.9.0/build/tools

8 Reference

1. A. Siegel, T. Tautges, A. Caceres, D. Kaushik, P. Fischer, G. Palmiotti, M.A. Smith, J. Ragusa, “Software Design of SHARP,” in *Proceedings of the Joint International Topical Meeting on Mathematics and Computations and Supercomputing in Nuclear Applications (M&C + SNA)*, American Nuclear Society, April 2007.
2. T.J. Tautges, R. Meyers, K. Merkley, C. Stimpson, C. Ernst, *MOAB: A Mesh-Oriented Database*, Sandia National Laboratories report SAND2004-1592, April 2004.
3. M.A. Smith, D. Kaushik, A. Wollaber, W.S. Yang, B. Smith, C. Rabiti, G. Palmiotti, “Recent Research Progress on PROTEUS at Argonne National Laboratory,” in *Proceedings of the International Conference on Mathematics, Computational Methods and Reactor Physics (M&C),* American Nuclear Society, April 2009.
4. P.F. Fischer, J.W. Lottes, S.G. Kerkemier, Nek5000 Web Page, http://nek5000.mcs.anl.gov, 2008.
5. D. Parsons, J.M. Solberg, R.M. Ferencz, M.A. Havstad, N.E. Hodge, and A.P. Wemhoff, *Diablo User Manual*, Lawrence Livermore National Laboratory report UCRL-SM-234927, Sept. 2007.
6. T.J. Tautges, H.-J. Kim, A. Caceres, R. Jain, “Coupled Multi-Physics simulation frameworks for reactor simulation: A Bottom-Up approach,” in *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C)*, American Nuclear Society, Rio de Janeiro, Brazil, May 2011.
7. D. Gaston, C. Newman, G. Hansen, D. Lebrun-Grandi, “MOOSE: a parallel computational framework for coupled systems of nonlinear equations,” *Nuclear Engineering and Design*, **239**(10):1768–1778, Oct. 2009.
8. D.E. Keyes et al., ”Multiphysics Simulations: Challenges and Opportunities,” *International Journal of High Performance Computing Applications*, **27**(1):4-83, 2012.
9. T. Tautges, P. Fischer, I. Grindeanu, R. Jain, V. Mahadevan, A. Obabko, M.A. Smith, E. Merzari, “SHARP Assembly-Scale Multiphysics Demonstration Simulations,” Technical Milestone Report ANL/NE-13/9, Argonne National Laboratory, 2013.
10. M.A. Smith, et al, “PROTEUS: development of a new reactor physics analysis tool,” in *Proceedings of Winter Meeting on International Conference on Making the Renaissance Real*, **97**:565–566, American Nuclear Society, Nov. 2007.
11. Y. Maday, A.T. Patera, “Spectral element methods for the Navier-Stokes equations,” in A.K. Noor and J.T. Oden, editors, *State-of-the-Art Surveys in Computational Mechanics*, pp. 71–143, ASME, New York, 1989.
12. A.G. Tomboulides, J.C.Y. Lee, and S.A. Orszag, “Numerical simulation of low Mach number reactive flows,” *Journal of Scientific Computing*, **12**:139–167, June 1997.
13. A.G. Tomboulides, M. Israeli, G.E. Karniadakis, “Efficient removal of boundary-divergence errors in time-splitting methods,” *Journal of Scientific Computing*, **4**:291–308, 1989.

14. G. Marleau, R. Roy, and A. Hébert, "DRAGON: A Collision Probability Transport Code for Cell and Supercell Calculations," Report IGE-157, Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, Québec, 1994.
15. Yiqi Yu, Elia Merzari, Aleksandr Obabko, Justin Thomas, A porous medium model for predicting the duct wall temperature of sodium fast reactor fuel assembly, Nuclear Engineering and Design, Volume 295, 15 December 2015, Pages 48-58,

Nuclear Engineering Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC